An ADL Perspective on Next Generation SCORM Requirements as Derived from Project Tin Can

Advanced Distributed Learning (ADL) Co-Laboratories

4 January 2012

This document was authored by Andy Johnson, a contractor with Problem Solutions in support of the Office of the Deputy Assistant Secretary of Defense (Readiness) Advanced Distributed Learning (ADL) Initiative. Please send all feedback and inquiries to the ADL Technical Team, via helpdesk@adlnet.gov

Introduction

The Sharable Content Object Reference Model (SCORM) has been a staple of online learning standards since 2001. ADL is now leading the effort of the Next Generation SCORM, which will update SCORM with Web 2.0 and beyond technologies. This effort is in a direct response to community feedback and user requirements determined over the lifetime of SCORM.

Rustici Software, in response to a Broad Area Announcement through ADL, collected a great deal of community feedback regarding the shortcomings of both the previous and current versions of the SCORM. This feedback took on multiple forms, the most significant being the data collected from the Tin Can User Voice site. This site collected open-ended community feedback, categorized it, and had participants "spend" voting points on which issues were most relevant to them. Ben Clark and Mike Rustici, each of whom has a decade of SCORM experience, facilitated this process.

This White Paper depicts the ADL position on next generation SCORM requirements based on the results to date from the Tin Can process and our own collective experience in interpreting community needs. As a contributor to each version of the SCORM since version 1.0, SCORM-tester, certifier, prototype developer, course architect, and author to hundreds of Help Desk tickets, I have a pretty good idea of what the shortcomings of SCORM are—both from a practical standpoint and from hearing the community voices. I couple my experiences with those recalled in conversations, emails, and presentations with others involved with the ADL program.

The following entries are in order as they were scored on the Tin Can User Voice site. These also constitute some recombination of issues that seemed to be similar or at least able to be categorized together. It also does not take into account the "beating a dead horse" effect, where some of the "more known" shortcomings of SCORM were not discussed to as great of length as some of the newer issues. In other words, the list isn't perfect, but gives some idea to what the concerns of the ADL Community are.

SCORM Should be Able to Handle Distributed Content

Not all content can be pre-packaged or even pre-determined as it seems the original SCORM expected. The concept of a package or .zip file is becoming more and more archaic. Content needs to be on distributed servers for a variety of reasons – from ownership issues to timely content updates. Technical issues, such as the cross-domain scripting problem, continue to throw wrenches in many content deployments that are not entirely contained within a content package.

Social media (and other forms of content that cannot be pre-loaded) is a growing trend of learning experiences and has no means of being tracked by the SCORM. Many learning experiences are not bundled into a .zip file and uploaded/accessed through a Learning Management System (LMS). These experiences need to be tracked in a way that is meaningful and that can integrate tracking with other formalized learning experiences traditionally found in LMSs. SCORM must adopt a model of streaming data that can allow tracking of these experiences and reporting of completion. A linking and bookmarking model has been suggested to bring more materials into the learning experience.

Furthermore, SCORM has always been a browser-based specification. Rich Internet Applications (RIAs) may not use browser technology, but still should be able to be tracked. The ideal fit would be if SCORM could act as a Web Service to enable it to integrate with all types of technology platforms as well as function outside large-scale deployments.

SCORM Needs to be Simpler (and Cheaper)!

SCORM has a built-in reliance on an LMS, which has two main drawbacks. First is the cost of implementation. An LMS requires too much overhead for the average small organization. There are open source possibilities, but many of them are cumbersome to implement and still require a dedicated person to manage the installation. The second drawback is a reliance on the suite of products included in an LMS. It must be weighed for all of its features. Chances are, an LMS will do at least one thing you do not want it to do. It is then a matter of picking the "lesser of all evils." It would be better if the Application Programming Interface (API) was better exposed and allowed customizable programming and separation of services. The Google API and Amazon APIs are two cited examples of such implementations.

While SCORM solves most of the use cases for which it was created, content creation is still difficult to do right out of the box. Currently, content developers are expected to read a 200-page technical specification before beginning. It was said best on the Tin Can User Voice site that a specification should "make a typical implementation easy and a complex implementation possible." It is important to realize that if content development requires a programmer to implement features without SCORM, it would probably still need one to implement those features with SCORM. Tools and templates would ease this process, but SCORM sequencing is still very difficult for tool vendors to implement effectively. Pre-requisites should be much simpler to specify than they are. The imsmanifest.xml is cumbersome and XML is too restrictive.

SCORM Should Handle Offline or Long-Running Content

Content failing because the API cannot find an LMS and error strings propagating when they cannot be used offline are examples of some of the struggles that learners and content developers face when deploying SCORM in a non-computer lab environment. Intermittent or disconnected content should be able to be tracked, especially given the rise in mobile devices that support online learning. State detection and resolution is currently beyond the scope of SCORM but it is a strongly desired feature. Theoretically an LMS could create a service to do this, but as mentioned, the cost and structure of an LMS is not a direction Next Generation SCORM should support. SCORM should have the tools to detect offline content and track the learner accordingly—from multiple platforms/devices.

Games and simulations, among others, can be long-running pieces of content that may not have defined boundaries as a SCO might. SCORM should support a more fluid approach in accessing, tracking, and experiencing content on or offline.

SCORM Should Provide a Way to Expose the Data it Tracks

One of the biggest disappointments of LMS implementations is when there is no interface provided to see the user data. SCORM requires a certain amount of data to be tracked. However, there is no requirement in SCORM that a user or programming interface exist to expose that data. It was assumed to be an LMS feature and beyond the scope of SCORM. Surprisingly, many LMSs do not expose the data to the level desired by content providers, administrators, teachers, and learners. It is not effective for an instructor to log into each individual student's account to see tracked data. A very typical use case is an instructor monitoring several students at once, seeing progress, and interacting with both students and SCORM tracking data in real time.

This is a sticky area however, knowing how far SCORM should go in providing exposure requirements, UI requirements, etc. SCORM could provide an API to directly access the data. It could also require an LMS to expose the data via user interfaces. A basic requirement is content that generates and publishes reports in a common format cross-SCO, cross-learner, and potentially cross-course (to handle entire curricula). Security, database issues, access control, and user privileges would still be handled by the LMS.

SCORM Should Ensure my "Sharable" Content is Interoperable and Portable

SCORM is supposed to enable interoperability. The issue is that interoperability at some levels is not enforced. It still occurs that one course designed to run in a particular LMS environment will not run in another. Many times the graphical issues include how an LMS implements navigation (SCORM only allows hiding/showing of controls), full screen mode, and launching of content (frame, window, etc.). These UI issues have always been left up to the developer and are beyond the scope of the current version of SCORM. Building a Table of Contents from a content package is another example of an interface construct that is not built uniformly across systems.

It has been suggested that SCORM provides an API for validation of a user interface. The implementation would be a set of tests to see if content really works in an LMS as the Conformance Test Suite seems to fall short of solving interoperability. There is no "reality check" with the Conformance Test Suite, just a series of outputs. There is also the issue that loading 100+ individual content packages and performing serial tests is extremely cumbersome.

With so many possible control options, there is blame to be shared between poor content design (designers bending navigation rules) and poor LMS interface design. Areas for which SCORM does not provide a recommendation—such as "what is a content package?", "what is a course?", "how should roll-up be used?", and "how should completion and success be used?"—have caused confusion and inconsistency. So too have the multiple conformance and certification levels and the lack of definition for tool certification.

SCORM Should Use Current Programming Standards

Programming has changed since the inception of SCORM in the late 1990s. Within SCORM, there has not been an update that reacted to the advancement of Web technology. SCORM needs to "get with the times" (warning: the rest of this section is filled with an alphabet soup of these technical advancements). SCORM is based on XML and JavaScript to be run in a web-browser via HTML. Newer web standards have become popular since the creation of SCORM. JSON, REST, and SOAP are newer web technologies that have benefits compared to the current technologies used in SCORM. It is recommended that SCORM use a RESTful API to decentralize the architecture. JSON can be used with JQuery to "get" and "set" the entire data model. The REST/SOAP model aids performance support. These solutions would solve cross-domain issues and reduce the need of direct synchronicity. Eliminating the construct of a "frame" would also open up possibilities to other types of content, including content developed for mobile devices, which do not support the HTML frame construct.

SCORM Should Move Beyond the Single Learner Approach

The SCORM has always been a single-user tracking model with information obscured from other users and even from SCO to SCO of the individual learner (the exception being the “data” field enabled in SCORM 2004 4th Edition). Extracting user data from the same SCO across multiple users has proven to be an area of concern. Concepts such as Team-based training, collaboration, sharing results, or even an awareness of others taking the same content with the ability to ask questions (peer-to-peer) outside the content were never included with SCORM. With movement towards team training, joint efforts, social media, and all other forms of collaboration, this is clearly an area that needs to expand. Also necessary would be customizations determined by an instructor. Access control, visibility of results, and other criteria may vary even across attempts of the same content.

SCORM Should Remove Sequencing Altogether

Sequencing has been a confusing part of the specification since its inception. SCORM Version 1.2 was very concise, compartmentalized, and comprehendible. Adding “Simple Sequencing” severely complicated every version of SCORM after 1.2. While the sequencing used in SCORM 2004 can do almost every use case imaginable, just getting a simple behavior off the ground can be quite cumbersome. A major reason that sharing doesn’t happen is because sequencing is confusing and the concept of SCO modularity is ambiguous. It is also the case that many developers chose to do sequencing internal to the content because doing it externally was extremely difficult. The ideal solution allows content to flow more seamlessly (as in one big SCO), but also be robust to track modularly (many small SCOs).

SCORM Should Have an Authentication Mechanism, Particularly to Protect Assessment Data

The biggest concern with bringing in distributed content, especially content objectives associated with completion/satisfaction, is the authoritative source to make sure that the distributed content and the instructor are on the same page. Does the content have the ability to set SCORM data values and thus determine pass/fail? Is instructor intervention required? Is LMS authentication required? A SCORM authoritative API could be necessary to validate the source of the content.

SCORM has always relied upon the authentication mechanism of the LMS. With movement away from this “all in one” reliance towards a service-based approach, a central means of authentication is necessary. It was suggested to use OAuth, a leading open protocol for protecting data, as a potential service or integration point.

Another issue with the nature of SCORM’s programming foundation and lack of authentication is that it has always been somewhat “hackable.” Learners can grab client-side code and often find correct answers to assessments within it. The Next Generation SCORM should have a means to disable this method of cheating.

SCORM Needs to Track More Robust Data

SCORM does not differentiate types of SCOs or even aggregations as “assessment” or “non-assessment,” which has been an unpopular gray area for content developers to figure out. It doesn’t contain IMS QTI or a similar integration for assessments, which trips up many LMSs. SCORM dangerously assumes that all content is graded "on the fly," the same for each student, and the same for each class. While SCORM doesn't particularly clash with existing specifications, integration has never been a project of ADL, nor has any specification been rolled into SCORM. The minimalist model of interactions is not robust enough to support the assessment needs of the community and should change with the Next Generation SCORM. SCORM, through APIs, needs to better support paragraph answers, simulation and game data, and content that might require instructor intervention. It should also be easily modified to support new types of interactions that do not currently exist.

SCORM imposes limits on the amount of tracked data. Unlike in the late 1990s, storage is cheap. Any amount of data can be tracked and kept. Programming should be expandable and not static. A "smallest permitted maximum" coupled with interoperability is essentially creating a cap on the amount of data. Also, the amount of time that data should be stored is partially ambiguous in SCORM and should be clarified. Which data is short-term and which is long-term? It has also been argued that all SCO attempts should be kept, not just the most recent. A creative use of "suspend" shouldn't be the means by which past data is tracked and used by content.

Conclusion

ADL is leading the effort to create the Next Generation SCORM. While a great deal of time and effort has been accomplished to gather requirements, the door is never closed on new requirements. Community participation is vital to create a more effective and useful learning/development environment. Please continue to follow the evolution of Next Generation SCORM on ADLnet.gov and offer your expertise. The past successes of SCORM can be attributed to the involvement of the ADL Community and their understanding of the process of building a specification. A specification isn't mandated, it is self-imposed and should be reliable. Let's all work together to build the Next Generation SCORM as a way to bring benefits to all of those who use distributed learning.

Glossary

API - Application Programming Interface – an agreed upon set of code that is intended to be used as an interface by software components to communicate with each other.

Cross-domain scripting – also called cross-site scripting. This is the practice of linking to content not within the same domain or hosted web area that the authorized content resides. This is considered a major security vulnerability and thus causes many languages, such as JavaScript (SCORM is essentially tied to JavaScript) to adopt a same origin policy for code. While more secure, it inhibits content that is not hosted in the same domain.

Google and Amazon APIs – Both of these are industry examples of how a well-used site can release a set of code that allows 3rd party deployed versions to tie-in to the same content, thus creating a similar look, feel, and functionality to the popular site. For example, the Amazon API has features allowing developers to advertise, let users search for, and aid in the discovery of Amazon products.

IMS QTI – The IMS Question and Test Interoperability (QTI) specification outlines a means of creating assessment information that allows questions, assessments and results to be shared across systems.

Rich Internet Application – An RIA is a Web application that takes a form similar to that of a desktop application. It is commonly delivered by a site-specific browser or through a browser plug-in. Adobe Flash, JavaFX, and Microsoft Silverlight are currently the three most common platforms.

Smallest Permitted Maximum – The smallest permitted maximum is a rule of the SCORM specifications regarding the number of data fields an LMS must create for a particular data model element. For example, if defined on the number of interactions as 256, an LMS must create space for at least 256 interactions for the content. It can choose to create more, but does not have to according to the specification.